基于哈希不透明谓词的 JavaScript 软件水印算法

吴伟民 林水明 余国鹏 林志毅
(广东工业大学计算机学院 广东 广州 510006)

摘 要 针对现有软件水印算法存在性能开销大或无法抵抗各类攻击的缺点和鲜有在 JavaScript 源码中实现的现状,提出一种基于哈希不透明谓词的 JavaScript 软件水印算法。该算法构造一种新的基于除留余数法哈希映射不透明谓词并将软件水印信息嵌入不透明谓词的表达式中,进而构造此不透明谓词的永假基本块嵌入程序中实现软件水印。开发了一个基于此算法的 JavaScript 软件水印系统。实验证明,该算法能在增加较少的系统开销的前提下有效抵抗各种常见的静动态攻击,同时还能提高水印的隐秘性和鲁棒性。

关键词 不透明谓词 除留余数法 软件水印 JavaScript

中图分类号 TP309.7 **文献标识码** A DOI:10.3969/j.issn.1000-386x.2016.04.071

A HASH OPAQUE PREDICATES-BASED SOFTWARE WATERMARKING ALGORITHM IN JAVASCRIPT

Wu Weimin Lin Shuiming Yu Guopeng Lin Zhiyi
(*Faculty of Computer, Guangdong University of Technology, Guangzhou 510006, Guangdong, China*)

Abstract Because of the weakness of current software watermarking algorithms in high performance overhead or suffering from various attacks, and of being scarcely implemented in JavaScript source code, we proposed a new hash opaque predicate-based JavaScript software watermarking algorithm. By constructing a new kind of hash-mapping opaque predicate, which is based on the method of remainder of division operation, and embedding the watermark information of software into opaque predicate expression, the algorithm constructs the everlasting basic block of this opaque predicate and embedding it into program to implement software watermarking. We also developed a JavaScript software watermarking system which is based on this algorithm. Experiments proved that the algorithm can effectively resist various common static and dynamic attacks on the premise of less system overhead increase, as well as improve the invisibility and robustness of the watermark.

Keywords Opaque predicate Method of remainder of division operation Software watermark JavaScript

0 引 言

软件水印是软件安全领域中的研究热点。其原理是通过借助数字水印的相关思想,将水印嵌入到软件中,以保护软件的知识产权和取缔各类非法盗版。目前研究的大部分软件水印技术都是基于 C/C ++ 或 Java 等编译型语言,对于诸如 JavaScript (JS)等解释型脚本语言的代码安全技术研究相对较少,而 JS 软件水印技术更加鲜见。随着云计算技术的快速发展,以 JS 为代表的客户端语言的使用更加广泛,如何保护 JS 版权是一项具有现实意义和经济效益的研究课题。

目前保护 JS 源码版权的主要研究有:Patrick 等人使用基于堆图的软件胎记技术来保护 JS 代码的安全性和版权[1],并通过构建对比树来提取软件胎记,而缺点是构建对比树的性能消耗很大,一般限制树的深度小于 3;Swati 进行改进提出基于凝聚聚类和频率子图的动态 JS 软件胎记技术[2],然而动态软件胎记存在只保护局部代码和不适合交互性软件的缺点。

在 C/C ++ 或 Java 软件水印领域的研究主要包括:Myles 和 Gregory 分别介绍了基于重新排列基本块[3]和在软件执行时重新分配寄存器[4]的软件水印算法,但此类算法的隐蔽性较差。为进一步提高水印的隐秘性,Nagra 从程序并发入手,提出基于线程关系的软件水印算法[5],许金超等对其进行了改进[6]。然而,以上方法的抗干扰性较差。Collberg 等人将扩频技术应用于软件水印中以提高抗干扰能力[7],但此方法是个非盲水印算法,且它嵌入的水印比特率相当低。汤战勇等对 PPCT 软件水印编码结构进行优化并根据特定策略进行加密,实现提高软件水印的隐秘性和防篡改[8],但此方法通过加密等操作增加系统开销,降低性能。同时,在软件水印中引入不透明谓词的研究有:Arboit 使用不透明谓词建立静态软件水印算法[9],Myles 对其进行动态扩展[10],然而此算法通过模式匹配的方法提取水印,无法抵抗模式匹配攻击;杨志刚将不透明谓词用于防篡改软件水印技术中能有效防止篡改[11],但在水印嵌入位置问题上缺

收稿日期:2014 - 12 - 09。广州市科技计划项目(2012Y2-00046, 2013Y2-00043);广东高校优秀青年创新人才培养计划项目(2012LYM_0054)。**吴伟民**,教授,主研领域:可视计算,系统工具与平台,代码安全。**林水明**,硕士生。**余国鹏**,本科生。**林志毅**,讲师。

乏讨论;高军提出基于中国剩余定理和拉格朗日插值公式的不透明谓词软件水印[12],但鲁棒性和隐蔽性较差。

为了进一步提高水印性能、降低系统开销和抵抗各类静动态攻击,本文提出一种基于哈希映射的改进构造不透明谓词的方法,并将此方法应用于 JS 软件水印中,提出一种基于不透明谓词的软件水印算法,同时对水印嵌入位置和水印内容进行讨论,并结合 JS 的特性开发一个基于此算法的 JS 软件水印系统。最后,将对算法性能、水印可靠性、隐秘性和抗干扰性等方面进行验证与分析。

1　不透明谓词的构造

不透明谓词就是谓词表达式在嵌入程序之前,其真值已经确定(通常为布尔类型),并广泛应用于分支结构和循环结构的判断条件中。目前构造不透明谓词的方法主要有:利用同余方程[9]和基于改进混沌映射 Logistic[13]的构造方法,而这些方法只能构造结果为布尔类型的不透明谓词。据此,本文提出一种能构造具有多种结果状态的不透明谓词的方法,此方法是基于哈希映射机制将输入从一数值空间映射成另一数值空间并输出,具体的算法过程描述如下:

Step1　确定映射机制集 $F = \{F_1, F_2, \cdots, F_N\}$,其中 $F_i(i = 1,2,\cdots,N)$ 必须满足以下条件:

i) $F_i(X) \rightarrow Y$。其中 $X \in \{X_1, X_2, \cdots, X_m\}$,$X_j = (x_1, x_2, \cdots, x_t)$ 是输入向量,当 $t = 1$ 时,F_i 是一元映射机制,$Y \in \{y_1, y_2, \cdots, y_n\}$ 是输出结果,也就是不透明谓词的状态集合。

ii) $F_i(i = 1,2,\cdots,N)$ 存在逆映射操作,即满足:$F_i^{-1}(Y) = X$,这个条件是实现构造不透明谓词的关键。

Step2　构造属于不透明谓词结果状态集合 $state = \{1,2,\cdots,n\}$ 的初始输入序列集 $X_{in} = (X_{in}^{state=1}, X_{in}^{state=2}, \cdots, X_{in}^{state=n})$,其中 $X_{in}^{state=j}$ 存放映射结果恒为 j 的输入序列。通过逆映射 $F_i^{-1}(Y) = X$ 实现:

i) 选择 $y_i \in Y$,根据逆映射 $F_i^{-1}(y_i) = X_{in}^{state=i}$,此时结果的个数可以是大于 1,逆映射是一种一对多的关系。

ii) 将结果存放回 X_{in},重新选择 y_i,重复 i)的操作直至产生达到上限数量的结果为止。

Step3　根据初始输入序列和映射机制构造不透明谓词 P。为输入序列添加伪随机性,根据随机函数 $random()$ 为每一种不透明谓词结果状态 y_i 选择一个初始序列 $x_{in}^{state=j}$。此时初始序列和映射机制构成产生不透明谓词 P 的秘钥 key,即 $key = (x_{in}^{state=1}, x_{in}^{state=2}, \cdots, x_{in}^{state=n}, F)$,$P = O(key)$。

本文以哈希函数构造方法除留余数法作为映射机制,实现构造具有 n 种状态的不透明谓词,具体的伪代码如图 1 所示。

```
Initial : p ← n;
          value ← 0;
          result[n];
// store Opaque Pridicate
loop : while(value < n)
         value ← value + 1;
         p ← p + 1; //adjust Parameters
         r ← int(random());
         input ← value + r * p;
       str ← input.tostring() + "mod p ";
        result[value] ← str;
endloop
output : result
```

图 1　基于除留余数法哈希函数的不透明谓词构造算法

为了提高不透明谓词抵抗逆向分析的能力,在具体实现中进行如下的优化:将输入序列存放在全局数组中,并通过引用的方式访问其中的元素[14];理论上可以使用所有满足条件的映射机制相结合,提高不透明谓词的复杂性;构造新型混合输入序列,由两个或两个以上的数组元素进行代数运算操作形成新的输入序列。

2　软件水印算法

2.1　基于 JS 的软件水印

软件水印技术已广泛应用于 C、C ++ 、Java 等流行语言编写的程序软件当中,然而对客户端语言诸如 JavaScript(JS)脚本程序的水印技术研究却很鲜见。JS 作为热门的脚本语言,广泛应用于各类 Web 应用开发当中,再者 JS 脚本嵌入在网页文件中,一般用户都可以使用浏览器获得源码,这对网站信息的安全性、源代码的保密性等都带来更大的挑战。因此,通过在 JS 代码中嵌入水印来保证软件著作权和防止盗版是一项具有现实意义和经济效益的研究课题。

2.2　不透明谓词软件水印算法

本文需要借助 Antlr 对 JS 程序源码进行语法分析并收集源码信息,包括源码的三类基本块:循环基本块(包括 while、for 和 do…while)、分支基本块(包括 if…else 和 switch…case)以及顺序基本块三类,基本块在运行中都是顺序执行的。据此,将上文构造的不透明谓词应用于 JS 软件水印当中,基于不透明谓词的 JS 软件水印算法包括水印的嵌入和水印的提取两个过程。

2.2.1　水印的嵌入算法

水印嵌入算法的过程描述如下:

Step1　初始化水印序列 M 和程序基本块数 N,其中 $M = m_1, m_2, \cdots, m_t$ 是长度为 t 的数字水印,并且 $m_i \in N^+$,$0 \leqslant m_i \leqslant 9$。

Step2　选择 M 中的一个位数 m_i,构造不透明谓词 $P(m_i)$,使得 $P(m_i) = m_i$。

Step3　构造永假基本块 FalseBlock,并且有:

$$FalseBlock = \begin{cases} \text{if } (P(m_i)\&\&ID_i)\{code;\} & m_i = 0 \\ \text{if } (!P(m_i)\&\&ID_i)\{code;\} & m_i \neq 0 \end{cases}$$

其中 if 结构可以替换成 while、do…while 或 for 结构,ID_i 用于指示 m_i 在 M 中的位置,code 部分是永不执行的代码,其选择方法主要有:

- 自定义生成:系统自动生成符合语法要求的垃圾代码。
- 源程序的任意函数:在程序代码中确定某一函数作为 FalseBlock 的代码部分。
- 随机选择源程序的函数:每次构造 FalseBlock 的同时随机确定某一函数做为代码部分。此方法的优点是:若代码中多次出现同一代码段容易引起攻击者的注意,并且即使某一水印片段的函数被破解,不会影响其他水印片段。

Step4　确定水印片段 m_i 嵌入的位置 p_i 并将 FalseBlock 嵌入在基本块 p_i 之后,并将 p_i 和唯一标识符 ID_i 保存到秘钥副本 $key(p)$ 当中用于水印提取。其嵌入的策略主要有:

- 简单策略。直接令 $p_i = Ni/t$,这是一种基于平均分配的思路,将水印片段均匀地分布在程序源码当中。
- 随机策略。包括可重复随机策略和不可重复随机策略:

a) 可重复随机策略,$p_i = random(1 \sim N)$。

b) 不可重复随机策略,$p_i = random(1 \sim N)$,$p_i \neq p_j (j = 1, 2, \cdots, i-1)$。

• 混合策略。在简单策略的基础上添加随机因子向量 r，$r=(r_1,r_2,\cdots,r_t)$，$r_i\in N^+$，$1\leqslant r_i\leqslant t$，$r_i=random(1\sim t)$，$r_i\neq r_j(j=1,2,\cdots,i-1;i=1,2,\cdots,t)$，简单地讲，$r$ 是正整数 $1\sim t$ 的不重复随机分布序列。此时水印片段的嵌入位置 $p_i=\frac{N}{t}r_i$，$p_i\neq p_j(j=1,2,\cdots,i-1)$。这种策略既能保证水印均匀分布，也能使得不透明谓词的位置具备随机性。

Step5 不断重复 Step2 – Step4 直至所有水印片段嵌入完成。

其原理如图 2 所示。

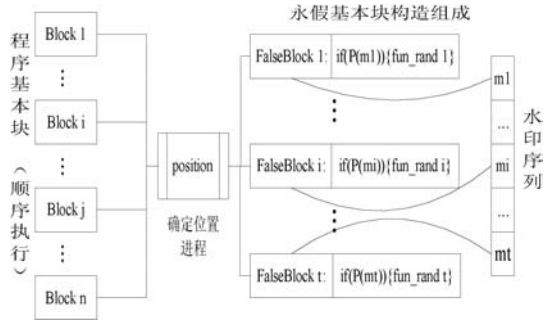

图 2 水印嵌入原理

2.2.2 水印的提取算法

水印提取算法的过程描述如下：

Step1 初始化已添加水印程序代码 $S(M)$，并对其进行源码分析，提取出顺序执行基本语句块 Block 1，…，Block n。

Step2 根据嵌入算法保存的水印秘钥副本获嵌入位置 p_i。

Step3 提取基本块 Block p_i 的首行 if 条件表达式并计算真值 m_i，并根据表达式中的 ID_i 将其保存在 M 中的相应位置。

Step4 不断重复 Step2 – Step3 直至所有嵌入位置完成计算。

水印的提取过程比较简单，关键是秘钥副本必须安全保存。提取过程原理如图 3 所示。

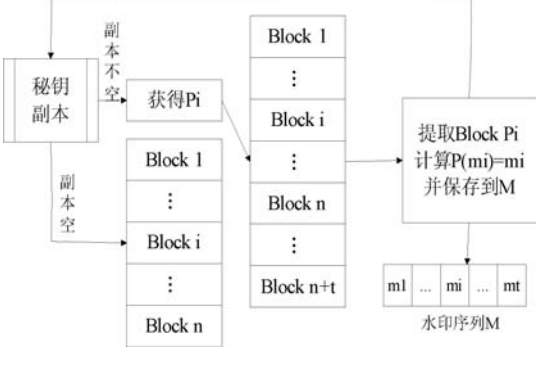

图 3 水印提取原理

3 实验结果及其分析

3.1 实验结果

借组 VS2010 C#语言集成开发环境，本文实现了一个基于不透明谓词的 JavaScript 脚本语言软件水印系统。系统包括随机生成水印序列、构造不透明谓词、不透明谓词水印的嵌入和提取等功能。使用 Google 公司开发的基准测试套件 V8 Benchmark Suit version7 作为测试用例，本系统成功实现对所有测试用例进行水印嵌入和提取，结果如图 4 – 图 6 所示。

图 4 不透明谓词生成结果

图 5 软件水印嵌入结果

图 6 软件水印提取结果

3.2 性能分析

负载性能 在程序中嵌入水印必定会增加系统的性能开销，主要包括程序变大和运行时间延迟。本文提出的基于不透明谓词软件水印算法通过添加永不运行的垃圾代码确保运行时间片变化不大，同时由于水印大小固定，因此本算法适合软件规模较大的情况。五个基准测试嵌入水印前后的容量、运行时间结果如表 1 所示。结果显示，水印嵌入前后测试用例在容量和运行时间的增长率仅为：4.92% 、8.70% 。

表 1 嵌入水印前后程序容量和运行时间对比

测试用例	程序容量		运行时间	
	水印前	水印后	水印前	水印后
richards. js	15.9 KB	16.8 KB	1.3 ms	1.5 ms
deltablue. js	25.0 KB	26.8 KB	2.1 ms	2.2 ms
crypto. js	46.8 KB	49.5 KB	5.0 ms	5.7 ms
regexp. js	108 KB	111 KB	5.8 ms	5.8 ms
earley-boyer. js	190 KB	196 KB	14.0 ms	15.3 ms

隐蔽性 分支语句是 JS 程序经常使用到的结构，通过将水

印片段隐藏在带不透明谓词的分支结构中不容易引起攻击者的注意,同时,构造分支语句的永假基本块都是原程序的部分函数代码,使得水印代码在编码风格、思维方式等方面都与程序的其他部分很相似。这样能在一定程度上提高水印的隐秘性。

鲁棒性　暴力破坏是相对于破解的另一种攻击行为,在保留软件功能的前提下修改程序代码、结构等操作获得相关权限,此时软件水印也有可能受到干扰。不透明谓词软件水印算法将水印分割成相互独立的片段并赋予唯一标识符,即使部分片段受到干扰,大部分信息仍能保留下来。并且水印提取算法是通过秘钥文件中的位置表进行水印检索和提取,能够有效抵制模式匹配攻击。如果通过非法修改顺序基本块的位置来破坏水印,程序将无法正确运行。通过对程序执行标识符模式匹配修改和打乱程序基本块攻击操作,程序执行结果和水印提取结果如表 2 所示。

表 2　模式匹配攻击和程序基本块攻击结果

攻击方式	水印提取结果	程序执行结果
模式匹配攻击	提取成功	运行成功
程序基本块攻击	提取失败	运行失败

3.3　安全性分析

3.3.1　静态攻击

静态分析技术主要有识别循环、别名分析、控制流分析、切片技术等。由于不透明谓词软件水印算法是通过将水印保存在分支表达式中,因此对程序循环进行识别不会影响水印的安全性;该算法将源数据保存在全局数组并通过引用访问数据,这样能有效抵抗别名分析[14];控制流分析和切片技术主要通过改变程序流程结构或者代码执行顺序来破坏水印完整性,而本文提出的水印算法是将水印信息隐藏在不透明谓词表达式中,水印信息并不依赖程序流程和执行顺序,也就是说表达式不被修改的前提下都能通过唯一标识符提取水印。通过对嵌入水印后程序进行控制流变形(如图 7 所示)再次进行水印提取,所有测试结果都能成功提取正确水印。

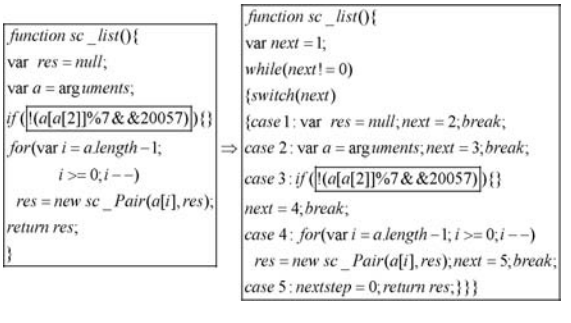

图 7　控制流压扁前后代码变形情况

3.3.2　动态攻击

攻击者还能通过设置断点进行调试、剖分和跟踪,进而分析程序执行路径和流程结构等信息,对于分支结构条件表达式,攻击者往往只注重条件的真假,忽视其中隐藏的信息,基于不透明谓词的软件水印算法能够逃避这类攻击。然而,反汇编工具能够优化删除程序不被执行的垃圾代码。据此,本文通过全局数组存放数据,并对数据进行运算以增强不透明谓词的隐蔽性,使得跟踪变得更加困难。同时,还可以将永假基本块改装成不影响程序结果但又被调用执行的代码,这样即使动态攻击也无法实现对软件水印的破坏。

4　结　语

本文提出一种基于不透明谓词的软件水印算法并开发一个基于此算法的 JS 脚本水印系统,通过实验证实了该算法能在增加较少的系统开销的前提下有效抵抗各种常见的静动态攻击,同时还能提高水印的隐秘性和鲁棒性。下一步的工作是优化和完善该算法,包括提高不透明谓词的隐秘性和扩展水印的内容。

参　考　文　献

[1] Chan P P F, Hui L C K, Yiu S M. Heap graph based software theft detection[J]. Information Forensics and Security, IEEE Transactions on, 2013, 8(1): 101 - 110.

[2] Patel, Swati J, Tareek M Pattewar. Software birthmark based theft detection of JavaScript programs using agglomerative clustering and Frequent Subgraph Mining[C]//Embedded Systems (ICES), 2014 International Conference on. IEEE, 2014.

[3] Ginger Myles, Christian Collberg, Zachary Heidepriem, et al. The evaluation of two software watermarking algorithms[J]. Software: Practice and Experience, 2005, 35(10): 923 - 938.

[4] Wolfe G, Wong J L, Potkonjak M. Watermarking graph partitioning solutions[J]. Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on, 2002, 21(10): 1196 - 1204.

[5] Nagra, Jasvir, Clark Thomborson. Threading software watermarks[C]// Berlin: Springer Heidelberg, 2005.

[6] 许金超,曾国荪. 一种基于线程关系的软件水印算法[J]. 电子学报, 2012, 40(5): 891 - 896.

[7] Christian Collberg, Tapas Ranjan Sahoo. Software watermarking in the frequency domain: implementation, analysis, and attacks[J]. Comput. Secur, 2005, 13(5): 721 - 755.

[8] 汤战勇,房鼎益,苏琳,等. 一种基于代码加密的防篡改软件水印方案[J]. 中国科学技术大学学报, 2011, 41(7): 599 - 606.

[9] Arboit, Genevieve. A method for watermarking java programs via opaque predicates[C]//The Fifth International Conference on Electronic Commerce Research (ICECR-5). 2002.

[10] Myles G, Collberg C. Software watermarking via opaque predicates: Implementation, analysis, and attacks[J]. Electronic Commerce Research, 2006, 6(2): 155 - 171.

[11] 杨志刚. 基于常量编码的防篡改软件水印技术[D]. 吉林:吉林大学, 2009.

[12] 高军. 软件水印算法研究[D]. 四川:电子科技大学, 2011.

[13] 苏庆,吴伟民,李忠良,等. 混沌不透明谓词在代码混淆中的研究与应用[J]. 计算机科学, 2013, 40(6): 155 - 160.

[14] Udupa Sharath K, Saumya K Debray, Matias Madou. Deobfuscation: Reverse engineering obfuscated code[C]//Reverse Engineering, 12th Working Conference on IEEE, 2005.

(上接第 305 页)

[12] Newman M E J. datsets. http://www - personal. umich. edu/ ~ mejn/netdata/.

[13] Watts D J, Strogatz S H. Collective dynamics of 'small-world' networks[J]. Nature, 1998, 393(6684): 440 - 442.

[14] Newman M E J, Park J. Why social networks are different from other types of networks[J]. Physical Review E, 2003, 68(3): 036122.